Variables

This chapter covers the following topics:

- Defining Your Own Variables
- Arithmetic Calculations
- Text Functions
- Decimal Numbers
- The Variable Character
- System Variables

Defining Your Own Variables

If a long word or phrase occurs frequently in the source text, you can assign it to a variable. Subsequently it is only necessary to specify the variable.

.SV - Set Variable

```
.SV variable-name=value
```

You can define a variable and then assign a value to it. For example:

```
.SV name=Smith
```

Each variable must be identified by a unique name. The variable name (the parameter before the equal sign) can only contain letters or digits. It must contain at least one character and can be up to 100 characters long. The variable name must not start with the variable character (see below) and must not contain blanks. No distinction is made between upper-case and lower-case.

The parameter after the equal sign can either be text (can consist of several words and may contain blanks) or an arithmetic calculation (see *Arithmetic Calculations with .SV*).

	Maximum value for text	**Maximum value for arithmetic calculation**
Parameter after equal sign	249 characters	100 characters
Intermediate result	CSIZE minus 10 (in KB)	31 digits
Final result	CSIZE minus 10 (in KB)	29 digits

Notes:

1. Starting with Con-form version 3.4.1, the maximum value for text may exceed 100 characters. In previous versions, this was restricted to 100 characters.
2. CSIZE is the size of Con-nect buffer area. The administrator defines it in the Natural parameter module NATPARM. The maximum CSIZE value can be 512 KB. Thus, the maximum that can be used for an intermediate and final text result is 502 KB.

No distinction is made between variables which hold text and variables which hold numeric values. A variable can even hold text at one instant and a numeric value at a later instant. The type of value which a variable currently holds is determined by the most recent .SV instruction which assigned a value to that variable.

Using the Variable in the Document Text

After you have defined a variable with the .SV instruction, you can include it in the document text. It must be preceded by the variable character. Initially, the variable character is the ampersand (&). For example:

```
&variable
```

When you format the document, the variable is replaced with the defined value.

Substitution (see the .SU instruction) is automatically switched on by the .SV instruction. This means that the text that is preceded by the variable character is interpreted as a variable and replaced accordingly.

Associating Text Strings and Variables

You can associate any number of text strings and variables.

To output a text string or another variable *after* the variable (without a blank in between), you must specify a period (.) directly after the first variable. The period is not required after a text string which is followed by a variable. For example:

```
.SV var1=butter
.SV var2=fly
.SV dollars=200
&var1.&var2
&var1.milk
dragon&var2
$&dollars
```

The above instructions cause the following formatted output:

```
butterfly
buttermilk
dragonfly
$200
```

Using Compound Names

A variable can be formed by resolving a compound name.

A compound name contains two or more simple variables and is resolved by substituting the value of each variable, working from right to left. For example:

```
.SV V1wakeup=Good Morning
.SV V2night=wakeup
.SV V3=night
&V1&V2&V3
```

The above instructions cause the following formatted output:

```
Good Morning
```

The variable in the above example is resolved from right to left in successive steps:

```
&V1&V2&V3
&V1&V2night
V1wakeup
Good Morning
```

Punctuation Mark After a Variable

To output a punctuation mark after the variable, you must enter a period (.) followed by the required punctuation mark. For example, to end a sentence with a period, you must repeat the period:

```
.SV text=In this case, you must repeat the period
&text..
.SV more=This is followed by a semicolon
&more.;
```

The above instructions cause the following formatted output:

```
In this case, you must repeat the period.
This is followed by a semicolon;
```

Arithmetic Calculations with .SV

The parameter of the .SV instruction can also be a numeric value. You can specify an integer, a fractional number with a decimal sign, or a number with a thousands separator character (see the .OP TRI instruction).

You can use the following arithmetic operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division

In the .SV instruction, a calculation is evaluated by applying the operators from left to right. All operators have equal priority. Parentheses are not evaluated.

Tip:
It is recommended that you use the .CV instruction for arithmetic operations since it evaluates parentheses. See *.CV - Compute Variable*.

You can specify fractional numbers in the .SV instruction. However, if you do not specify the number of characters which are to be output after the decimal character (see the .OP DAS instruction), the result is rounded to the nearest integer. For example:

```
.SV number1=2.5*3+2
The result of number1 is &number1
.OP DAS=2
.SV number2=2.5*3+2
The result of number2 is &number2
```

The above instructions cause the following formatted output:

```
The result of number1 is 9
The result of number2 is 9.50
```

Caution:
You must specify the .OP DAS instruction before the .SV instruction. Otherwise, the result is not rounded as desired.

You can modify the value of a parameter. For example:

```
.SV counter=11
.SV counter=&counter+1
&counter
```

The above instructions cause the following formatted output:

```
12
```

Initially, the result of a division is rounded to the nearest integer. However, when you specify the number of characters which are to be output after the decimal character (see the .OP DAS instruction), the result is rounded accordingly. For example:

```
.OP DAS=2
.SV number=2/3
&number
```

The above instructions cause the following formatted output:

```
0.67
```

Arithmetic Calculations

It is recommended that you use the .CV instruction for arithmetic operations since it evaluates parentheses (in contrast to the .SV instruction).

.CV - Compute Variable

```
.CV variable-name=arithmetic-expression
```

You can define a variable and then compute its value using an arithmetic expression. An arithmetic expression consists of one or more constants, variables or system variables.

A constant is an integer, a fractional number with a decimal sign, or a number with a thousands separator character (see the .OP TRI instruction). Arithmetic operators (see below) must not be used as thousands separator characters. The number of digits in the constant must not exceed 29. The number of digits after the decimal sign must not exceed 7. For example:

```
.OP TRI=’,’
.CV const1=15
.CV const2=0.123
.CV const3=1234567890123456789012.1234567
.CV const4=1,999
```

A variable used in an arithmetic expression must have a numeric value. This variable must have been defined in a previous .SV or .CV instruction.

A system variable can be used in an arithmetic expression if it has a numeric value.

You can use the following arithmetic operators:

( )	Parentheses
*	Multiplication
/	Division
+	Addition
-	Subtraction

Parentheses are evaluated. The arithmetic operation is processed in the following order:

1. Parentheses
2. Multiplication and division (left to right)
3. Addition and subtraction (left to right)

When the divisor is 0, the result of a division operation is 0.

When the result of a .CV calculation leads to an overflow, the variable is set to 5 asterisks (*****). For example:

```
.CV var1=-25000
&var1
.CV var2=15-5+4/2-10
&var2
.CV var3=(1+2) * 3/ (1+8)
&var3
.CV var4=&var1+100000
&var4
.CV var5= 1000000000000000 * 1000000000000000
.IF &var5 = *****
.TH
Your result is too big
.EL
&var5
.EI
```

The above instructions cause the following formatted output:

```
-25000
2
1
75000
Your result is too big
```

Precision of Results for Arithmetic Operations

The following table gives an overview of the precision rules that are used in an arithmetic calculation with the .CV instruction.

Operation	**Digits before decimal character**	**Digits after decimal character (intermediate result)**	**Digits after decimal character (final result)**
Addition/Subtraction	Fi + 1 or Si + 1 (whichever is greater)	Fd or Sd (whichever is greater, maximum 7)	DAS and RND are applied to the last intermediate result.
Multiplication	Fi + Si + 2	Fd + Sd (maximum 7)	
Division	Fi + Sd	If .OP REM=OFF: Fd or value of DAS option (whichever is greater, maximum 7). In addition, if .OP RND=ON, this number of digits is internally increased (+1).	
		If .OP REM=ON: value of DAS option. This value is also used for remainder.	

The following abbreviations are used in the above table:

F	First operand.
S	Second operand.
i	Digits before decimal character.
d	Digits after decimal character.

The number of digits after the decimal character is handled separately for intermediate and final result. Intermediate results are arithmetic operations *before* rounding and applying of the option DAS. The final result is last intermediate result *after* rounding and applying of the option DAS.

In the final result (formatted output), the digits after the decimal character are defined by the option DAS.

When the option RND is switched ON, rounding is performed. When the option RND is switched OFF, the last digits are truncated.

When the option REM is switched ON, the result of a division operation is not rounded (regardless of the setting defined for .OP RND).

Example:

```
.OP DAS=0
.OP RND=ON
.CV  VAR=(3+5.25) / 1.5 - 4
```

The final result of the above operation is 2, as indicated in the following table:

	Operation	**Result**
Intermediate operation 1	3+5.25	8.25
Intermediate operation 2	8.25/1.5	5.500 (since .OP RND=ON, the number of digits after decimal character is internally increased (+1)).
Intermediate operation 3	5.500-4	1.500
Final operation	Apply DAS and RND	2

.OP RND - Round Result of .CV

```
.OP RND=ON
.OP RND=OFF
```

This instruction can only be used with the .CV instruction.

Initially, the result of the .CV instruction is rounded to the nearest integer. This corresponds to the following instruction:

```
.OP RND=ON
```

For example:

```
.OP DAS=3
.OP RND=ON
.CV var1=0 + 0.2555
&var1
.OP RND=OFF
.CV var2=0 + 0.2555
&var2
```

The above instructions cause the following formatted output:

```
0.256
0.255
```

.OP REM - Remainder of Division with .CV

```
.OP REM=ON
.OP REM=OFF
```

This instruction can only be used with the .CV instruction.

Initially, this option is switched off. This corresponds to the following instruction:

```
.OP REM=OFF
```

If the remainder of a division operation is to be moved to the modifiable system variable $RR, you must specify the following instruction:

```
.OP REM=ON
```

The initial value of $RR is 0 (zero).

The result of a division operation is not rounded (regardless of the setting defined for .OP RND). The format of the remainder depends on the options defined with the instructions .OP DAS and .OP TRI.

For example:

```
.OP DAS=2
.OP REM=ON
&$RR
.CV number=2/3
&number
&$RR
```

The above instructions cause the following formatted output:

```
0
0.66
0.02
```

Text Functions

You can use text functions in conjunction with the .SV or .CV instruction. You must always specify an apostrophe (’) after the function code.

Text functions can also be used with the instructions .IF and .WH.

The following function codes are available:

Code	Explanation	Example

E	Check whether the specified variable exists. The parameter is the name of a variable. If the variable exists, the value 1 is returned. If the variable does not exist, the value 0 is returned.	`.SV var=Smith`
		`.SV test=var`
		`If variable "var" exists,`
		`type 1,`
		`else type 0:`
		`&E’&test`

H	Convert text string to hexadecimal value. The parameter can be an arbitrary text string. Each character in the parameter string is converted to the corresponding EBCDIC hexadecimal value.	`.SV var=H’123`
		`The hexadecimal value of 123`
		`is &var.`
		`.SV name=Smith`
		`The hexadecimal value of Smith`
		`is &H’&name.`

I	Invert rendition of text string. The parameter is an arbitrary text string. The direction in which the contents of the variable is interpreted is altered.	`.SV abc=xyz`
		`.SV def=I’&abc`
		`The inverted value is`
		`&def (zyx).`

L	Convert text string to lower-case. The parameter can be an arbitrary text string. Each upper-case letter in the parameter string is replaced by the lower-case equivalent. Other characters are left unchanged.	`.SV var=CATS`
		`Don’t forget to feed your`
		`&L’&var tonight.`

N	Determine length of text string. The parameter can be an arbitrary text string. The number of characters in the parameter string is returned.	`.SV date=3.11.00`
		`The length of this string`
		`is &N’&date`

R	Convert parameter to a Roman number. The parameter is an unsigned decimal number. The number is returned as the corresponding Roman number in upper-case format.	`.SV year=R’2000`
		`&year will be a wonderful year.`
		`.SV vol=8`
		`Give me volume &R’&vol of the`
		`encyclopaedia.`

U	Convert text string to upper-case. The parameter is an arbitrary text string. Each lower-case letter in the parameter string is replaced by the upper-case equivalent. Other characters are left unchanged.	`.SV var=Smith`
		`Please call Mr. &U’&var`
		`before you go home.`

X	Convert hexadecimal value to a character. The parameter must be a hexadecimal value. The hexadecimal value is converted to its one-character long equivalent. You can specify only one hexadecimal value as the parameter.	`.SV hex=C5`
		`The hexadecimal value C5`
		`denotes &X’&hex`
		`.SV hexa=X’C5`
		`The hexadecimal value C5`
		`denotes &hexa`

Specifying the Text Function as a Parameter

You can specify a text function as a parameter of the .SV or .CV instruction. To do so, you must specify the required function code followed by the parameter. You can then include the variable in the running text; it must be preceded by the variable character. Initially, the variable character is the ampersand (&). For example:

```
.SV var=H’123
The hexadecimal value of 123 is &var
```

The above instructions cause the following formatted output:

```
The hexadecimal value of 123 is F1F2F3
```

Specifying the Text Function in the Running Text

When you use the text function in the running text, the function code must be preceded by the variable character. Furthermore, you must specify a defined variable after the text function. The variable must also be preceded by the variable character. Initially, the variable character is the ampersand (&). For example:

```
.SV var=123
The hexadecimal value of 123 is &H’&var
```

The above instructions cause the following formatted output:

```
The hexadecimal value of 123 is F1F2F3
```

Decimal Numbers

You can use the following options to specify how decimal numbers are to be output when arithmetic calculations are included in variables.

.OP DAS - Number of Characters After the Decimal Character

```
.OP DAS=number
```

Initially, the result of an arithmetic calculation is rounded to the nearest integer. To avoid this, you must specify the number of characters which are to be output after the decimal character. The decimal character is defined using the .OP DEC instruction.

For example, if 2 characters are to be output after the decimal character, you must specify:

```
.OP DAS=2
```

You can specify a maximum of 7 characters after the decimal character.

The decimal character is *not* inserted automatically when specifying the .SV or .CV instruction as in the following example:

```
.OP DAS=2
.SV number=1
&number
```

The above instructions cause the following formatted output:

```
1
```

To output the above variable with a decimal character, you should perform the following additional calculation step:

```
.OP DAS=2
.SV number=1+0
&number
```

The above instructions cause the following formatted output:

```
1.00
```

.OP DEC - A Different Decimal Character

```
.OP DEC=character
```

Initially, the Natural decimal character is used. By default, this is the period (.). You can define a different decimal character. For example, to define the comma as the new decimal character, you must specify:

```
.OP DEC=’,’
```

Since the comma is used by certain Con-form instructions to separate parameters, it is important that you enclose it in apostrophes as shown above. If you want to define, for example, the slash (/), you need not use the apostrophes.

.OP TRI - Thousands Separator Character

```
.OP TRI=character
.OP TRI=ON
.OP TRI=OFF
```

Initially, a thousands separator character is not defined.

You can specify the character that is to be used as the thousands separator character. For example, to define the comma as the thousands separator character, you must specify:

```
.OP TRI=’,’
```

Since the comma is used by certain Con-form instructions to separate parameters, it is important that you enclose it in apostrophes as shown above. If you want to define, for example, the slash (/), you need not use the apostrophes.

The thousands separator character is inserted in the formatted output as the result of an arithmetic calculation. It is *not* inserted when specifying the .SV or .CV instruction as in the following example:

```
.OP TRI=’,’
.SV number=2000
&number
```

The above instructions cause the following formatted output:

```
2000
```

To output the above variable with a thousands separator character, you should perform the following additional calculation step:

```
.OP TRI=’,’
.SV number=2000+0
&number
```

The above instructions cause the following formatted output:

```
2,000
```

If you no longer want to use the thousands separator character, you can deactivate it using the following instruction:

```
.OP TRI=OFF
```

If you want to reuse the previously defined, deactivated thousands separator character, you can activate it using the following instruction:

```
.OP TRI=ON
```

If you specify .OP TRI=ON and a thousands separator character has not yet been defined, the comma is used by default.

The Variable Character

All variables in the document text (i.e. the variables you defined using the .SV or .CV instruction as well as the system variables) must be preceded by the variable character. If required, you can define another variable character or switch the recognition of the variable character off.

.OP VSG - A Different Variable Character

```
.OP VSG=character
```

This instruction defines the character that is used to distinguish text and variables.

Initially, the variable character is the ampersand (&). However, you can define a different character. For example, to define the paragraph sign (§) as the variable character, you must specify:

```
.OP VSG=§
```

It is not possible to use the Dollar sign ($) as the variable character for system variables. Thus, it is not possible to specify, for example, $$PL. However, it is possible to define the following:

```
.SV PL=&$PL
.OP VSG=$
$PL
```

.SU - Substitution

```
.SU ON
.SU OFF
```

When substitution is switched on, each string in the source text which is preceded by the variable character is interpreted as a variable and replaced accordingly.

Initially, substitution is switched off. However, it is automatically switched on by the .SV or .CV instruction. This corresponds to the following:

```
.SU ON
```

Substitution is also switched on by macro calls *with* parameters. It is not switched on by macro calls without parameters.

You can switch the recognition of the variable character off. This is necessary if the source text contains strings that include the variable character and you do not want them to be interpreted as variables.

To switch substitution off and thus cancel the effect of the variable character, you must specify:

```
.SU OFF
```

System Variables

In addition to the variables you define yourself (using the .SV or .CV instruction), you can also use system variables in your source text. System variables can contain values such as the current date and time, or the current text margins.

When you specify a system variable, it must be preceded by the variable character. Initially, the variable character is the ampersand (&). For example, to use the system variable $DT, you must specify it as follows:

```
&$DT
```

In addition to inserting a variable in the running text, you can also use it with the instructions .IF and .WH.

Fixed System Variables

Fixed system variables cannot be modified. Con-form automatically replaces the variables below with their actual values when the document is formatted.

Variable	Explanation	Value
$CN	Century (the first two digits of the year).	19 or 20
$DA	Day.	1...31
$DD	Day name in Danish.	Mandag...Søndag
$DF	Day name in French.	Lundi...Dimanche
$DG	Day name in German.	Montag...Sonntag
$DN	Day name in English.	Monday...Sunday
$DT	Date. For the months 1 through 9 a blank is inserted before the number of the month (for example, 9. 1.93).	dd.mm.yy
$DY	Julian date.	1...366
$HO	Hour.	00...23
$MD	Month name in Danish.	Januar...December
$MF	Month name in French.	Janvier...Décembre
$MG	Month name in German.	Januar...Dezember
$MI	Minute.	00...59
$MN	Month name in English.	January...December
$MO	Month number.	1...12
$SE	Second.	00...59
$YE	Year.	00...99

The following example illustrates how to use fixed system variables:

```
The current date is: &$DN., &$MN &$DA., &$CN.&$YE..
```

The above instructions cause the following formatted output:

```
The current date is: Wednesday, October 25, 2000.
```

Modifiable System Variables

The following variables can be used in complex formatting situations. Initially, they are set to default values. However, when one of the instructions shown in the right column is issued, the value of the variable is modified.

Variable	Explanation	Modified by
$BM	Bottom margin.	.BM
$CH	Chapter number.	.CH, .SC
$FM	Footer margin.	.FM
$FN	Footnote counter. The variable $FN is incremented every time it is referenced. It is used for consecutive numbering of footnotes.	
$FS	Footer space.	.FS
$HM	Header margin.	.HM
$HS	Header space.	.HS
$IN	Indentation.	.LM, .OF, .TI, .CS
$IX	Last index entry.	.IX
$LC	Remaining lines on page.	.SV, .LS
$LL	Line length.	.LL, .CS
$PL	Page length.	.PL, .LS
$PN	Current page number. When the variable $PN is replaced in your text, Arabic page numbering is used. With the .SV or .CV instruction and in the top and bottom titles, the page-number character can be used instead of $PN. In this case, the page number is always output according to your specifications (either Arabic or Roman numbers). Initially, the page-number character is the hash (#).	.NP, .PN
$RM	Right margin.	.RM, .CS
$RR	Remainder of division when .OP REM=ON.	.CV
$TM	Top margin.	.TM

The above modifiable system variables return values which have been internally saved by Con-form. However, it is not guaranteed that the system variables always produce the same results in different environments and with different versions of Con-form.

The following example illustrates how to use modifiable system variables. It also introduces the .IF instruction.

```
When less than 5 lines are available on the current page, the box is output on
the next page. The current page number is output in the box.
.SL 1
.IF &$LC < 5;.NP
.BX 10,50
.LM 10;.RM 49
.IL 1;.CE 1
This box is printed on Page &$PN..
.IL 1;.BX OFF
```

The above instructions cause the following formatted output:

```
When less than 5 lines are available on the current page, the box is output on
the next page. The current page number is output in the box.

         +---------------------------------------+
         !                                       !
         !    This box is printed on Page 1.     !
         !                                       !
         +---------------------------------------+
```